

BAM format specification for PacBio

The BAM format is a binary, compressed, record-oriented container format for raw or aligned sequence reads. The associated SAM format is a text representation of the same data. The [specifications for BAM/SAM](#) are maintained by the SAM/BAM Format Specification Working Group.

PacBio-produced BAM files are fully compatible with the BAM specification. In this document we describe the way we make use of the extensibility mechanisms of the BAM specification to encode PacBio-specific information, as well as conventions we adhere to.

An example file adhering to this specification will be maintained in the *pbcore* Python library.

Version

The PacBio BAM specification version described here is 3.0.5. PacBio BAM files adhering to this spec contain the tag `pb:3.0.5` in the `@HD` header.

Coordinate conventions

The BAM format uses a 0-based coordinate system to refer to positions and intervals on the reference.

PacBio also uses a 0-based coordinate system to refer to positions and intervals within sequence reads. Positions in PacBio reads are reckoned from the first ZMW read base (as base 0), *not* the first base in the HQ region.

Perhaps confusingly, the text SAM format uses 1-based coordinate system.

Note that following the SAM/BAM specification, 0-based coordinate intervals are defined as half-open (end exclusive) while 1-based intervals are closed.

Query versus aligned query terminology

A sequence read presented to an aligner is termed a *query*; typically this query will be a subsequence of an entire PacBio ZMW read—most commonly, it will be a *subread*, which is basecalls from a single pass of the insert DNA molecule.

Upon alignment, generally only a subsequence of the query will align to the reference genome, and that subsequence is referred to as the *aligned query*. Under *soft-clipping*, the entirety of the query is stored in the aligned BAM, but the CIGAR field indicates that some bases at either end are excluded from the alignment.

Abstractly, we denote the extent of the *query* in ZMW read as $[qStart, qEnd)$ and the extent of the aligned subinterval as $[aStart, aEnd)$. The following graphic illustrates these intervals:



In our BAM files, the `qStart`, `qEnd` are contained in the `qs` and `qe` tags, (and reflected in the `QNAME`); the bounds of the *aligned query* in the ZMW read can be determined by adjusting `qs` and `qe` by the number of soft-clipped bases at the ends of the alignment (as found in the CIGAR).

Note: In the legacy `cmp.h5` file format, soft-clipping was not possible, and the bounds of the original query were not stored. Only `aStart`, `aEnd` were stored, although in that file format they were referred to as `rStart`, `rEnd`.

QNAME convention

By convention the `QNAME` (“query template name”) for unrolled reads and subreads is in the following format:

```
{movieName}/{holeNumber}/{qStart}_{qEnd}
```

where `[qStart, qEnd]` is the 0-based coordinate interval representing the span of the *query* in the ZMW read, as above.

For CCS reads, the `QNAME` convention is:

```
{movieName}/{holeNumber}/ccs
```

CIGAR conventions

The “M” CIGAR op (`BAM_CMATCH`) is *forbidden* in PacBio BAM files. PacBio BAM files use the more explicit ops “X” (`BAM_CDIFF`) and “=” (`BAM_CEQUAL`). PacBio software will abort if `BAM_CMATCH` is found in a CIGAR field.

BAM filename conventions

Since we will be using BAM format for different kinds of data, we will use a `suffix.bam` filename convention:

Data type	Filename template
ZMW reads from movie	<i>movieName.zmws.bam</i>
Analysis-ready subreads ¹ from movie	<i>movieName.subreads.bam</i>
Excised adapters, barcodes, and rejected subreads	<i>movieName.scraps.bam</i>
CCS reads computed from movie	<i>movieName.ccs.bam</i>
Aligned subreads in a job	<i>jobID.aligned_subreads.bam</i>
Aligned CCS in a job	<i>jobID.aligned_ccs.bam</i>

1

Data in a `subreads.bam` file should be *analysis ready*, meaning that all of the data present is expected to be useful for downstream analyses. Any subreads for which we have strong evidence will not be useful (e.g. double-adaptor inserts, single-molecule artifacts) should be excluded from this file and placed in `scraps.bam` as a `Filtered` with an `SC` tag of `F`.

BAM sorting conventions

Aligned PacBio BAM files shall be sorted by position in the standard fashion as done by `samtools sort`. The `BAM @HD::SO` tag shall be set to `coordinate`.

Unaligned PacBio BAM files shall be sorted by `QNAME`, so that all subreads from a ZMW hole are stored contiguously in a file, with groups by ZMW hole number in numerical order, and within a ZMW, numerically by `qStart`. In case subreads and CCS reads are combined in a BAM, the CCS reads will sort after the subreads (`ccs` follows `{qStart}_{qEnd}`). Note that this sorting is not strictly alphabetical, so we shall set the `BAM @HD::SO` tag to `unknown`.

Use of headers for file-level information

Beyond the usual information encoded in headers that is called for SAM/BAM spec, we encode special information as follows.

`@RG` (read group) header entries:

`ID` tag (identifier)

contains an 8-character string interpretable as the hexadecimal representation of an integer. Read groups should have distinct `ID` values.

 v: 5.1 ▾

Note: Read group identifiers for PacBio data are calculated as follows:

```
RGID_STRING := md5(movieName + "/" + readType)[:8]
RGID_INT    := int32.Parse(RGID_STRING)
```

where *movieName* is the moviename (@RG::PU) and *readType* is the read type (found in @RG::DS). Note that *movieName* is lowercase while *readType* is uppercase. *md5* is understood to be the (lowercase) hex md5 digest of the input string.

RGID_STRING is used in the @RG header and in the RG tag of BAM records, while RGID_INT is used in the PacBio BAM index file.

Note that RGID_INT may be negative.

Example: CCS reads for a movie named “movie32” would have

- RGID_STRING = “f5b4ffb6”
- RGID_INT = -172687434

PL tag (“platform”):

contains “PACBIO”

PM tag (“platform model”)

contains “ASTRO”, “RS”, or “SEQUEL”, reflecting the PacBio instrument series

PU tag (“platform unit”):

contains the PacBio movie name.

DS tag (“description”):

contains some semantic information about the reads in the group, encoded as a semicolon-delimited list of “Key=Value” strings, as follows:

Mandatory items:

Key	Value spec	Value example
READTYPE	One of ZMW, HQREGION, SUBREAD, CCS, SCRAP, or UNKNOWN	SUBREAD
BINDINGKIT	Binding kit part number	100236500
SEQUENCINGKIT	Sequencing kit part number	001558034
BASECALLERVERSION	Basecaller version number	2.1
FRAMERATEHZ	Frame rate in Hz	100
CONTROL	TRUE if reads are classified as spike-in controls, otherwise CONTROL key is absent	TRUE

Note: The READTYPE values encountered in secondary analysis will be limited to SUBREAD and CCS. The remaining READTYPE values will only be encountered in intermediate steps before secondary analysis.

Base feature manifest—absent item means feature absent from reads:

Key	Value spec	Value example
DeletionQV	Name of tag used for DeletionQV	dq
DeletionTag	Name of tag used for DeletionTag	dt
InsertionQV	Name of tag used for InsertionQV	iq
MergeQV	Name of tag used for MergeQV	mq
SubstitutionQV	Name of tag used for SubstitutionQV	sq
SubstitutionTag	Name of tag used for SubstitutionTag	st
lpd:Frames	Name of tag used for IPD, in raw frame count.	ip
lpd:CodecV1	Name of tag used for IPD, compressed according to Codec V1.	ip
PulseWidth:Frames	Name of tag used for PulseWidth, in raw frame count.	pw

 v: 5.1 ▾

Key	Value spec	Value example
PulseWidth:CodecV1	Name of tag used for PulseWidth, compressed according to Codec V1.	pw

Optional items:

Note: These items are optional if there are no “bc” tags in the reads belonging to this read-group, otherwise they are mandatory.

Key	Value spec	Value example
BarcodeFile	Name of the Fasta file containing the sequences of the barcodes used	pacbio_384_barcodes.fasta
BarcodeHash	The MD5 hash of the contents of the barcoding sequence file, as generated by the <i>md5sum</i> commandline tool	0a294bb959fc6c766967fc8beeb4d88d
BarcodeCount	The number of barcode sequences in the Barcode File	384
BarcodeMode	Experimental design of the barcodes Must be Symmetric/Asymmetric/Tailed or None	Symmetric
BarcodeQuality	The type of value encoded by the bq tag Must be Score/Probability/None	Probability

Use of read tags for per-read information

Tag	Type	Description
qs	i	0-based start of query in the ZMW read (absent in CCS)
qe	i	0-based end of query in the ZMW read (absent in CCS)
zm	i	ZMW hole number
np	i	NumPasses (1 for subreads, variable for CCS—encodes number of <i>complete</i> passes of the insert)
rq	f	Float in [0, 1] encoding expected accuracy
sn	B,f	4 floats for the average signal-to-noise ratio of A, C, G, and T (in that order) over the HQRegion

Use of read tags for per-read-base information

The following read tags encode features measured/calculated per-basecall. Unlike `SEQ` and `QUAL`, aligners will not orient these tags. They will be maintained in *native* orientation (in the same order and sense as collected from the instrument) even if the read record has been aligned to the reverse strand.

Tag	Type	Description
dq	Z	DeletionQV
dt	Z	DeletionTag
iq	Z	InsertionQV
mq	Z	MergeQV
sq	Z	SubstitutionQV
st	Z	SubstitutionTag
ip	B,C or B,S	IPD (raw frames or codec V1)
pw	B,C or B,S	PulseWidth (raw frames or codec V1)

Notes:

- QV metrics are ASCII+33 encoded as strings
- *DeletionTag* and *SubstitutionTag* represent alternate basecalls, or “N” when there is no alternate basecall available. In other words, they are strings over the alphabet “ACGTN”.
- Encoding of kinetics features (`ip`, `pw`) is described below.

How to annotate scrap reads

Reads that belong to a read group with READTYPE=SCRAP have to be annotated in a hierarchical fashion:

1. Classification with tag `sz` occurs on a per ZMW level, distinguishing between spike-in controls, sentinels of the basecaller, malformed ZMWs, and user-defined templates.
2. A region-wise annotation with tag `sc` to label adapters, barcodes, low-quality regions, and filtered subreads.

Tag	Type	Description
<code>sz</code>	A	ZMW classification annotation, one of N:=Normal, C:=Control, M:=Malformed, or S:=Sentinel ¹
<code>sc</code>	A	Scrap region-type annotation, one of A:=Adapter, B:=Barcode, L:=LQRegion, or F:=Filtered ²

1

reads in the subreads/hqregions/zmws.bam file are implicitly marked as Normal, as they stem from user-defined templates.

2

`sc` tags 'A', 'B', and 'L' denote specific classes of non-subread data, whereas the 'F' tag is reserved for subreads that are undesirable for downstream analysis, e.g., being artifactual or too short.

QUAL

The `QUAL` field in BAM alignments is intended to reflect the reliability of a basecall, using the Phred-encoding convention, as described in the [SAM spec](#).

Both CCS and raw read BAM files respect this convention; historically, and for the present moment, the encoded probability reflects the confidence of a basecall against alternatives including substitution, deletion, and insertion.

We expect that more details will follow here in a later spec revision.

Subread local context

Some algorithms can make use of knowledge that a subread was flanked on both sides by adapter or barcode hits, or that the subread was in one orientation or the other (as can be deduced when asymmetric adapters or barcodes are used).

To facilitate such algorithms, we furnish the `cx` bitmask tag for subread records. The `cx` value is calculated by binary OR-ing together values from this flags enum:

```
enum LocalContextFlags
{
    ADAPTER_BEFORE = 1,
    ADAPTER_AFTER  = 2,
    BARCODE_BEFORE = 4,
    BARCODE_AFTER  = 8,
    FORWARD_PASS   = 16,
    REVERSE_PASS   = 32
};
```

Orientation of a subread (designated by one of the mutually exclusive `FORWARD_PASS` or `REVERSE_PASS` bits) can be reckoned only if either the adapters or barcode design is asymmetric, otherwise these flags must be left unset. The convention for what is considered a “forward” or “reverse” pass is determined by a per-ZMW convention, defining one element of the asymmetric barcode/adaptor pair as the “front” and the other as the “back”. It is up to tools producing the BAM to determine whether to use adapters or barcodes to reckon the orientation, but if pass directions cannot be confidently and consistently assessed for the subreads from a ZMW, neither orientation flag should be set. Tools consuming the BAM should be aware that orientation information may be unavailable for subreads in a ZMW, but if it is available for any subread in the ZMW, it will be available for all subreads in the ZMW.

The `ADAPTER_*` and `BARCODE_*` flags reflect whether the subread is flanked by adapters or barcodes at the ends.

This tag is mandatory for subread records, but will be absent from non-subread records (scraps, ZMW read, CCS read, etc.)

Tag	Type	Description
<code>cx</code>	i	Subread local context Flags

 v: 5.1 ▾

Barcode analysis

In multiplexed workflows, we record per-subread tags representing the barcode call and a score representing the confidence of that call. The actual data used to inform the barcode calls—the barcode sequences and associated pulse features—will be retained in the associated `scraps.bam` file, so that `bam2bam` can be used at a later time to reconstitute the full-length ZMW reads in order, for example, to repeat barcode calling with different options.

Tag	Type	Description
<code>bc</code>	B,S	Barcode Calls (per-ZMW)
<code>bq</code>	i	Barcode Quality (per-ZMW)

- Both the `bc` and `bq` tags are calculated `per-ZMW`, so every subread belonging to a given ZMW should share identical `bc` and `bq` values. The tags are also inter-depended, so if a subread has the `bc` tag, it must also have a `bq` tag and vice-versa. If the tags are present for any subread in a ZMW, they must be present for all of them. In the absence of barcodes, both the `bc` and `bq` tags will be absent
- The `bc` tag contains the *barcode call*, a `uint16[2]` representing the inferred forward and reverse barcode sequences (as determined by their ordering in the Barcode FASTA), or more succinctly, it contains the integer pair B_F, B_R . Integer codes represent 0-based position in the FASTA file of barcodes.
- The integer (`int`) `bq` tag contains the barcode call confidence. If the `BarcodeQuality` element of the header is set to `Score`, then the tag represents the mean normalized sum of the calculated Smith-Waterman scores that support the call in the `bc` tag across all subreads. For each barcode, the sum of the Smith-Waterman score is normalized by the length of the barcode times the match score, then multiplied by 100 and rounded; this provides an integer value between 0 - 100. On the other hand, if the value of the header-tag is `Probability` instead, then the tag value is a the Phred-scaled posterior probability that the barcode call in `bc` is correct. In both cases, the value will never exceed the `int8` range, but for backward-compatibility reasons we keep the BAM `bq` as `int`. This contract allows the PBI to store `bq` as a much smaller `int8`.

Barcode information will follow the same convention in CCS output (`ccs.bam` files).

Examples (subreads)

Scenario	<code>bc</code>	<code>bq</code>	<code>cx</code>
No barcodes, end-to-end, unknown orientation	<i>absent</i>	<i>absent</i>	<code>1 2 = 3</code>
Asymmetric barcodes, end-to-end, forward pass	<code>{ 1, 37 }</code>	<code>35</code>	<code>1 2 4 8 16 = 31</code>
Symmetric barcodes, end-to-end	<code>{ 8, 8 }</code>	<code>33</code>	<code>1 2 4 8 = 15</code>
Barcoded, HQ region terminates before second barcode; unknown orientation	<code>{ 8, 8 }</code>	<code>33</code>	<code>1 4 = 5</code>

Alignment: the contract for a mapper

An aligner is expected to accept BAM input and produce aligned BAM output, where each aligned BAM record in the output preserves intact all tags present in the original record. The aligner should not attempt to orient or complement any of the tags.



(Note that this contrasts with the handling of `SEQ` and `QUAL`, which are mandated by the BAM/SAM specification to be (respectively) reverse-complemented, and reversed, for reverse strand alignments.)

Alignment: soft-clipping

In the standard production configuration, PacBio's aligners will be used to align either subreads or CCS reads. In either case, we will use *soft clipping* to preserve the unaligned bases at either end of the query in the aligned BAM file.

Encoding of kinetics pulse features

Interpulse duration (IPD) and pulsewidth are measured in frames; natively they are recorded as a `uint16` per pulse/base event. They may be encoded in BAM read tags in one of two fashions:

- losslessly as an array of `uint16`; necessary for PacBio-internal applications but entails greater disk space usage.
- lossy 8-bit compression stored as a `uint8` array, following the codec specified below ("codec V1"). Provides a substar  v: 5.1  disk-space savings without affecting important production use cases (base modification detection).

In the default production instrument configuration, the lossy encoding will be used. The instrument can be switched into a mode (PacBio-internal mode) where it will emit the full lossless kinetic features.

The lossy encoding for IPD and pulsewidth values into the available 256 codepoints is as follows (**codec v1**):

Frames	Encoding
0 .. 63	0, 1, .. 63
64, 66, .. 190	64, 65, .. 127
192, 196 .. 444	128, 129 .. 191
448, 456, .. 952	192, 193 .. 255

In other words, we use the first 64 codepoints to encode frame counts at single frame resolution, the next 64 to encode the frame counts at two-frame resolution, and so on. Durations exceeding 952 frames are capped at 952. Durations not enumerated in “Frames” above are rounded to the nearest enumerated duration then encoded. For example, a duration of 194 frames would round to 196 and then be encoded as codepoint 129.

This encoding has the following features, considered essential for internal analysis use cases:

- *Exact* frame-level resolution for small durations (up to 64 frames)
- Maximal representable duration is 9.52 seconds (at 100fps), which is reasonably far into the tail of the distributions of these metrics. Analyses of “pausing” phenomena may still need to account for this censoring.

A reference implementation of this encoding/decoding scheme can be found in *pbcore*.

Unresolved issues

- Need to move from strings to proper array types for QVs
- ‘/’ preferable to ‘:’ in “IPD:CodecV1”
- Desire for spec for shorter movienames, especially if these are ending up in QNAMEs.